L Number	Hits	Search Text	DB	Time stamp
1	4427	(garbage near2 collect$4)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/15 12:21
2	18	((garbage near2 collect$4)) and multiprocessor and (synchroniz$6 with processors!)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/15 12:24
3	39	((garbage near2 collect$4)) and (multiprocessor or processors! or multi-processor or ((multiple or plurality or number or multi$6) adj2 processor)) and (synchroniz$6 with processors!)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/15 12:23
4	33	(((garbage near2 collect$4)) and (multiprocessor or processors! or multi-processor or ((multiple or plurality or number or multi$6) adj2 processor)) and (synchroniz$6 with processors!)) and synchroniz$6 near9 processor	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/15 12:24

Citation

Conference on High Performance Networking and Computing >archive
Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM) >toc
1997 , San Jose, CA

A scalable mark-sweep garbage collector on large-scale shared-memory machines

Authors
Toshio Endo The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan
Kenjiro Taura The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan
Akinori Yonezawa The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

Sponsors
IEEE-CS\DATC : IEEE Computer Society
SIGARCH : ACM Special Interest Group on Computer Architecture

Publisher
ACM Press New York, NY, USA

Pages: 1 - 14 Series-Proceeding-Article
Year of Publication: 1997
ISBN:0-89791-985-8

doi> http://doi.acm.org/10.1145/509593.509641 (Use this link to Bookmark this page)

> full text > abstract > references > citings > index terms > peer to peer

> Discuss > Similar > Review this Article Save to Binder

> BibTex Format

↑ **FULL TEXT:** Access Rules

pdf 97 KB

↑ **ABSTRACT**

This work describes implementation of a mark-sweep garbage collector (GC) for shared-memory machines and reports its performance. It is a simple "parallel" collector in which all processors cooperatively traverse objects in the global shared heap. The collector stops the application program during a collection and assumes a uniform access cost to all locations in the shared heap.

h c g c cf c

Implementation is based on the Boehm-Demers-Weiser conservative GC (Boehm GC). Experiments have been done on Ultra Enterprise 10000 (Ultra Sparc processor 250 MHz, 64 processors). We wrote two applications, BH (an N-body problem solver) and CKY (a context free grammar parser) in a parallel extension to C++.Through the experiments, We observe that load balancing is the key to achieving scalability. A naive collector without load redistribution hardly exhibits speed-up (at most fourfold speed-up on 64 processors). Performance can be improved by dynamic load balancing, which exchanges objects to be scanned between processors, but we still observe that straightforward implementation severely limits performance. First, large objects become a source of significant load imbalance, because the unit of load redistribution is a single object. Performance is improved by splitting a large object into small pieces before pushing it onto the mark stack. Next, processors spend a significant amount of time uselessly because of serializing method for termination detection using a shared counter. This problem suddenly appeared on more than 32 processors. By implementing non-serializing method for termination detection, the idle time is eliminated and performance is improved. With all these careful implementation, we achieved average speed-up of 28.0 in BH and 28.6 in CKY on 64 processors.

↑ REFERENCES

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

1 Josh Barnes and Piet Hut. A hirarchical O(N log N) force-calculation algorithm. Nature, 324:446-449, 1986.

2 Hans-Juergen Boehm, Space efficient conservative garbage collection, Proceedings of the conference on Programming language design and implementation, p.197-206, June 21-25, 1993, Albuquerque, New Mexico, United States

3 Hans-Juergen Boehm , Mark Weiser, Garbage collection in an uncooperative environment, Software—Practice & Experience, v.18 n.9, p.807-820, September 1988

4 A. A. Chien, U. S. Reddy, J. Plevyak, and J. Dolby. ICC++ - A C++ dialect for high performance parallel computing. In Proceedings of the 2nd International Symposium on Object Technologies for Advanced Software, volume 1049 of Lecture Notes in Computer Science, pages 76-95, 1996.

5 David L. Detlefs. Concurrent garbage collection for C++. In Topics in Advanced Language Implementation, chapter 5, pages 101-134. The MIT Press, 1991.

6 Damien Doligez , Xavier Leroy, A concurrent, generational garbage collector for a multithreaded implementation of ML, Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, p.113-123, March 1993, Charleston, South Carolina, United States

7 Robert H. Halstead, Jr., MULTILISP: a language for concurrent symbolic computation, ACM Transactions on Programming Languages and Systems (TOPLAS), v.7 n.4, p.501-538, Oct. 1985

8 Maurice Herlihy, A methodology for implementing highly concurrent data objects, ACM Transactions on Programming Languages and Systems (TOPLAS), v.15 n.5, p.745-770, Nov. 1993

9 Maurice P. Herlithy and J. Eliot B. Moss. Lock-free garbage collection for multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 3(3):304-311, May 1992.

10 Lorenz Huelsbergen , James R. Larus, A concurrent copying garbage collector for languages that distinguish (im)mutable data, Proceedings of the Fourth ACM SIGPLAN symposium on Principles & practice of parallel programming, p.73-82, May 19-22, 1993, San Diego, California,

United States

11 Nobuyuki Ichiyoshi and Masao Morita. A shared-memory parallel extension of KLIC and its garbage collection. In Proceedings of FGCS '94 Workshop on Parallel Logic Programming, pages 113-126, 1994.

12 Akira Imai and Evan Tick. Evaluation of parallel copying garbage collection on a shared-memory multiprocessor. IEEE Transactions on Parallel and Distributed Systems, 4(9):1030-1040, September 1993.

13 Richard Jones , Rafael Lins, Garbage collection: algorithms for automatic dynamic memory management, John Wiley & Sons, Inc., New York, NY, 1996

14 Yoshikazu Kamoshida. HOCS: A C++ extension with parallel objects and multi-threading. February 1997. (senior thesis), The University of Tokyo.

15 J. S. Miller , B. S. Epstein, Garbage collection in MultiScheme, Proceedings of the US/Japan workshop on Parallel Lisp on Parallel Lisp: languages and systems, p.138-160, June 1990, Sendai, Japan

16 James O'Toole , Scott Nettles, Concurrent replicating garbage collection, Proceedings of the 1994 ACM conference on LISP and functional programming, p.34-42, June 27-29, 1994, Orlando, Florida, United States

17 Joseph P. Skudlarek, Remarks on A methodology for implementing highly concurrent data, ACM SIGPLAN Notices, v.29 n.12, p.87-93, Dec. 1994

18 Joseph P. Skudlarek, Notes on "A methodology for implementing highly concurrent data objects", ACM Transactions on Programming Languages and Systems (TOPLAS), v.17 n.1, p.45-46, Jan. 1995

19 Kenjiro Taura, Satoshi Matsuoka, and Akinori Yonezawa. ABCL/f: A future-based polymorphic typed concurrent object-oriented language - its design and implementation -. In number 18 in Dimacs Series in Discrete Mathematics and Theoretical Computer Science, pages 275-292. the DIMACS work shop on Specification of Algorithms, 1994.

20 Kenjiro Taura , Akinori Yonezawa, An effective garbage collection strategy for parallel programming languages on large scale distributed-memory machines, Proceedings of the sixth ACM SIGPLAN symposium on Principles & practice of parallel programming, p.264-275, June 18-21, 1997, Las Vegas, Nevada, United States

21 Paul R. Wilson. Uniprocessor garbage collection techniques. In Proceedings of the 1992 SIGPLAN International Workshop on Memory Management, pages 1-42, 1992.

↑ CITINGS *4*

Guy E. Blelloch , Perry Cheng, On bounding time and space for multiprocessor garbage collection, ACM SIGPLAN Notices, v.34 n.5, p.104-117, May 1999

Toshio Endo , Kenjiro Taura, Reducing pause time of conservative collectors, Proceedings of the third international symposium on Memory management, June 20-21, 2002, Berlin, Germany

Yoav Ossia , Ori Ben-Yitzhak , Irit Goft , Elliot K. Kolodner , Victor Leikehman , Avi Owshanko, A parallel, incremental and concurrent GC for servers, ACM SIGPLAN Notices, v.37 n.5, May 2002

Yossi Levanoni , Erez Petrank, An on-the-fly reference counting garbage collector for Java, ACM SIGPLAN Notices, v.36 n.11, p.367-380, 11/01/2001

↑ INDEX TERMS

Keywords:
dynamic load balancing, garbage collection, parallel algorithm, scalability, shared-memory machine

↑ Peer to Peer - Readers of this Article have also read:

- Constructing reality
  Proceedings of the 11th annual international conference on Systems documentation
  Douglas A. Powell , Norman R. Ball , Mansel W. Griffiths
- Toward a real-time Ada design methodology
  Proceedings of the conference on TRI-ADA '90
  Norman R. Howes
- Fashioning conceptual constructs in Ada
  Proceedings of the conference on TRI-ADA '90
  Robert C. Shock
- Reuse: the two concurrent life cycles paradigm
  Proceedings of the conference on TRI-ADA '90
  Richard Drake , William Ett
- An experiment with Graphite
  Proceedings of the conference on TRI-ADA '90
  John Chludzinski , Robert Chi Tau Lai

h c g c cf c

Citation

International Conference on Supercomputing >archive
Proceedings of the 5th international conference on Supercomputing >toc
1991 , Cologne, West Germany

A space-efficient parallel garbage compaction algorithm

Author
Wolfgang Küchlin

Sponsor
SIGARCH : ACM Special Interest Group on Computer Architecture

Publisher
ACM Press New York, NY, USA

Pages: 40 - 46 Series-Proceeding-Article
Year of Publication: 1991
ISBN:0-89791-434-1

doi> http://doi.acm.org/10.1145/109025.109039 (Use this link to Bookmark this page)

↑ **REFERENCES**

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

CM88 K. Mani Chandy, Parallel program design: a foundation, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1988

Cra88 J. Crammond, A garbage collection algorithm for shared memory parallel processors, International Journal of Parallel Programming, v.17 n.6, p.497-522, Dec. 1989

DLM+78 Edsger W. Dijkstra , Leslie Lamport , A. J. Martin , C. S. Scholten , E. F. M. Steffens, On-the-fly garbage collection: an exercise in cooperation, Communications of the ACM, v.21 n.11, p.966-975, Nov. 1978

FY69 Robert R. Fenichel , Jerome C. Yochelson, A LISP garbage-collector for virtual-memory computer systems, Communications of the ACM, v.12 n.11, p.611-612, Nov. 1969

Hal85 Robert H. Halstead, Jr., MULTILISP: a language for concurrent symbolic computation, ACM Transactions on Programming Languages and Systems (TOPLAS), v.7 n.4, p.501-538, Oct. 1985

KN91 Wolfgang W. Kiichlin and Nicholas J. Nevin. On multi-threaded list-processing and garbage collection. Technical Report OSU-CISRC-3/91- TR11, Computer and Information Science Research Center, The Ohio State University, Columbus, OH 43210-1277, March 1991.

Knu73 Donald E. Knuth, The art of computer programming, volume 1 (3rd ed.): fundamental algorithms, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, 1997

Küc90 Wolfgang W. KSchlin. PARSAC-2: A parallel SAC-2 based on threads. In Proc. AAECC-8, LNCS, Tokyo, Japan~ August 1990. Springer- Verlag. To appear.

ME90 J. S. Miller , B. S. Epstein, Garbage collection in MultiScheme, Proceedings of the US/Japan workshop on Parallel Lisp on Parallel Lisp: languages and systems, p.138-160, June 1990, Sendai, Japan

Mer85 Susan M. Merritt, An inverted taxonomy of sorting algorithms, Communications of the ACM, v.28 n.1, p.96-99, Jan. 1985

Mor78 F. Lockwood Morris, A time- and space-efficient garbage compaction algorithm, Communications of the ACM, v.21 n.8, p.662-665, Aug. 1978

Ste75 Guy L. Steele, Jr., Multiprocessing compactifying garbage collection, Communications of the ACM, v.18 n.9, p.495-508, Sept. 1975

↑ INDEX TERMS

Primary Classification:

D. Software

↳ **D.4** OPERATING SYSTEMS

↳ **D.4.2** Storage Management

↳ **Subjects:** Allocation/deallocation strategies

Additional Classification:

E. Data

↳ **E.4** CODING AND INFORMATION THEORY

↳ **Subjects:** Data compaction and compression

F. Theory of Computation

↳ **F.2** ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY

General Terms:

h c g c cf c

Algorithms

↑ **Peer to Peer - Readers of this Article have also read:**

- Editorial pointers
  Communications of the ACM 44, 9
  Diane Crawford

- News track
  Communications of the ACM 44, 9
  Robert Fox

- Forum
  Communications of the ACM 44, 9
  Diane Crawford

- Representing extended entity-relationship structures in relational databases: a modular approach
  ACM Transactions on Database Systems (TODS) 17, 3
  Victor M. Markowitz , Arie Shoshani

- Editorial
  interactions 8, 5
  Steven Pemberton

h c g c cf c

Search Results

Search Results for: **[heaps <near/9> memory<AND>((synchronizing <near/9> processors<AND>((multiprocessor and synchronization and garbage and heaps))))]**
Found **16** of **131,734 searched.**

Search within Results

GO > Advanced Search

> Search Help/Tips

Sort by: **Title** **Publication** **Publication Date** **Score** Binder

Results 1 - 16 of 16 **short listing**

1 Portable, unobtrusive garbage collection for multiprocessor systems 94%
Damien Doligez , Georges Gonthier
Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages February 1994
We describe and prove the correctness of a new concurrent mark-and-sweep garbage collection algorithm. This algorithm derives from the classical on-the-fly algorithm from Dijkstra et al. [9]. A distinguishing feature of our algorithm is that it supports multiprocessor environments where the registers of running processes are not readily accessible, without imposing any overhead on the elementary operations of loading a register or reading or initializing a field. Furthermor ...

2 Real-time concurrent collection on stock multiprocessors 89%
A. W. Appel , J. R. Ellis , K. Li
ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation June 1988
Volume 23 Issue 7
We've designed and implemented a copying garbage-collection algorithm that is efficient, real-time, concurrent, runs on commercial uniprocessors and shared-memory multiprocessors, and requires no change to compilers. The algorithm uses standard virtual-memory hardware to detect references to "from space" objects and to synchronize the collector and mutator threads. We've implemented and measured a prototype running on SRC's 5-processor Firefly. It will be straightforward to merg ...

3 Mostly lock-free malloc 85%
Dave Dice , Alex Garthwaite
ACM SIGPLAN Notices , Proceedings of the third international symposium on Memory management June 2002
Volume 38 Issue 2 supplement
Modern multithreaded applications, such as application servers and database engines,

can severely stress the performance of user-level memory allocators like the ubiquitous malloc subsystem. Such allocators can prove to be a major scalability impediment for the applications that use them, particularly for applications with large numbers of threads running on high-order multiprocessor systems.This paper introduces Multi-Processor Restartable Critical Sections, or MP-RCS. MP-RCS permits user-level ...

4 Garbage collection for strongly-typed languages using run-time type reconstruction 84%

Shail Aditya , Christine H. Flood , James E. Hicks

ACM SIGPLAN Lisp Pointers , Proceedings of the 1994 ACM conference on LISP and functional programming July 1994

Volume VII Issue 3

Garbage collectors perform two functions: live-object detection and dead-object reclamation. In this paper, we present a new technique for live-object detection based on run-time type reconstruction for a strongly typed, polymorphic language. This scheme uses compile-time type information together with the run-time tree of activation frames to determine the exact type of every object participating in the computation. These reconstructed types are then used ...

5 A scalable mark-sweep garbage collector on large-scale shared-memory machines 83%

Toshio Endo , Kenjiro Taura , Akinori Yonezawa

Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM) November 1997

This work describes implementation of a mark-sweep garbage collector (GC) for shared-memory machines and reports its performance. It is a simple "parallel" collector in which all processors cooperatively traverse objects in the global shared heap. The collector stops the application program during a collection and assumes a uniform access cost to all locations in the shared heap. Implementation is based on the Boehm-Demers-Weiser conservative GC (Boehm GC). Experiments have been done on Ultra ...

6 Real-time, concurrent checkpoint for parallel programs 82%

K. Li , J. F. Naughton , J. S. Plank

ACM SIGPLAN Notices , Proceedings of the second ACM SIGPLAN symposium on Principles & practice of parallel programming February 1990

Volume 25 Issue 3

We have developed and implemented a checkpointing and restart algorithm for parallel programs running on commercial uniprocessors and shared-memory multiprocessors. The algorithm runs concurrently with the target program, interrupts the target program for small, fixed amounts of time and is transparent to the checkpointed program and its compiler. The algorithm achieves its efficiency through a novel use of address translation hardware that allows the most time-consuming operations of the c ...

7 On performance and space usage improvements for parallelized compiled APL code 82%

Dz-ching Ju , Wai-Mee Ching , Chuan-lin Wu

ACM SIGAPL APL Quote Quad , Proceedings of the international conference on APL '91 July 1991

Volume 21 Issue 4

Loop combination has been a traditional optimization technique employed in APL

compilers, but may introduce dependencies into the combined loop. We propose an analysis method by which the compiler can keep track of the change of the parallelism when combining high-level primitives. The analysis is necessary when the compiler needs to decide a trade-off between more parallelism and a further combination. We also show how the space usage, as well as the performance, improves by using system calls ...

8 Core semantics of multithreaded Java 80%

Jeremy Manson , William Pugh

Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande June 2001

Java has integrated multithreading to a far greater extent than most programming languages. It is also one of the only languages that specifies and requires safety guarantees for improperly synchronized programs. It turns out that understanding these issues is far more subtle and difficult than was previously thought. The existing specification makes guarantees that prohibit standard and proposed compiler optimizations; it also omits guarantees that are necessary for safe execution of much ex ...

9 Hoard: a scalable memory allocator for multithreaded applications 80%

Emery D. Berger , Kathryn S. McKinley , Robert D. Blumofe , Paul R. Wilson

Proceedings of the ninth international conference on Architectural support for programming languages and operating systems November 2000

Volume 28 , 34 Issue 5 , 5

Parallel, multithreaded C and C++ programs such as web servers, database managers, news servers, and scientific applications are becoming increasingly prevalent. For these applications, the memory allocator is often a bottleneck that severely limits program performance and scalability on multiprocessor systems. Previous allocators suffer from problems that include poor performance and scalability, and heap organizations that introduce false sharing. Worse, many allocators exhibit a dramatic incr ...

10 Hoard: a scalable memory allocator for multithreaded applications 80%

Emery D. Berger , Kathryn S. McKinley , Robert D. Blumofe , Paul R. Wilson

ACM SIGPLAN Notices November 2000

Volume 35 Issue 11

Parallel, multithreaded C and C++ programs such as web servers, database managers, news servers, and scientific applications are becoming increasingly prevalent. For these applications, the memory allocator is often a bottleneck that severely limits program performance and scalability on multiprocessor systems. Previous allocators suffer from problems that include poor performance and scalability, and heap organizations that introduce false sharing. Worse, many allocators exhibit a dramatic incr ...

11 GUM: a portable parallel implementation of Haskell 80%

P. W. Trinder , K. Hammond , J. S. Mattson , A. S. Partridge , S. L. Peyton Jones

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1996 conference on Programming language design and implementation May 1996

Volume 31 Issue 5

GUM is a portable, parallel implementation of the Haskell functional language. Despite sustained research interest in parallel functional programming, GUM is one of the first such systems to be made publicly available.GUM is message-based, and portability is facilitated by using the PVM communications harness that is available on many multi-

processors. As a result, GUM is available for both shared-memory (Sun SPARCserver multiprocessors) and distributed-memory (networks of workstations) architec ...

12 Cost-effective object space management for hardware-assisted real-time garbage collection 80%

Kelvin D. Nilsen , William J. Schmidt

ACM Letters on Programming Languages and Systems (LOPLAS) December 1992 Volume 1 Issue 4

Modern object-oriented languages and programming paradigms require finer-grain division of memory than is provided by traditional paging and segmentation systems. This paper describes the design of an OSM (Object Space Manager) that allows partitioning of real memory on object, rather than page, boundaries. The time required by the OSM to create an object, or to find the beginning of an object given a pointer to any location within it, is approximately one memory cycle. Object sizes are lim ...

13 Mul-T: a high-performance parallel Lisp 80%

D. A. Kranz , R. H. Halstead , E. Mohr

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation June 1989 Volume 24 Issue 7

Mul-T is a parallel Lisp system, based on Multilisp's future construct, that has been developed to run on an Encore Multimax multiprocessor. Mul-T is an extended version of the Yale T system and uses the T system's ORBIT compiler to achieve "production quality" performance on stock hardware — about 100 times faster than Multilisp. Mul-T shows that futures can be implemented cheaply enough to be useful in a production-quality system. Mul-T is fully operational, including a ...

14 An implementation of portable standard LISP on the BBN butterfly 77%

Mark Swanson , Robert Kessler , Gary Lindstrom

Proceedings of the 1988 ACM conference on LISP and functional programming January 1988

An implementation of the Portable Standard Lisp (PSL) on the BBN Butterfly is described. Butterfly PSL is identical, syntactically and semantically, to implementations of PSL currently available on the VAX, Gould, and many 68000-based machines, except for the differences discussed in this paper. The differences include the addition of the future and touch constructs for explicit parallelism and an extension of the fluid binding mechanism to support the mult ...

15 Fine-grain multithreading with minimal compiler support—a cost effective approach to implementing efficient multithreading languages 77%

Kenjiro Taura , Akinori Yonezawa

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation May 1997 Volume 32 Issue 5

It is difficult to map the execution model of multithreading languages (languages which support fine-grain dynamic thread creation) onto the single stack execution model of C. Consequently, previous work on efficient multithreading uses elaborate frame formats and allocation strategy, with compilers customized for them. This paper presents an alternative cost-effective implementation strategy for multithreading languages which can maximally exploit current sequential C compilers. We identify a s ...

16 A space-efficient parallel garbage compaction algorithm 77%
Wolfgang Küchlin
Proceedings of the 5th international conference on Supercomputing June 1991

Results 1 - 16 of 16 **short listing**

h c g e cf c

Search Results

Search Results for: **[processors and heaps and memory and sychronization]**
Found **2** of **131,734 searched.**

Search within Results

GO > Advanced Search

> Search Help/Tips

Sort by: **Title** **Publication** **Publication Date** **Score** Binder

Results 1 - 2 of 2 **short listing**

1 Session 20: software performance: Parallel performance prediction using lost cycles analysis 77%

Mark E. Crovella , Thomas J. LeBlanc

Proceedings of the 1994 ACM/IEEE conference on Supercomputing November 1994

Most performance debugging and tuning of parallel programs is based on the "measure-modify" approach, which is heavily dependent on detailed measurements of programs during execution. This approach is extremely time-consuming and does not lend itself to predicting performance under varying conditions. Analytic modeling and scalability analysis provide predictive power, but are not widely used in practice, due primarily to their emphasis on asymptotic behavior and the difficulty of developing acc ...

2 Parallel execution for serial simulators 77%

David Nicol , Philip Heidelberger

ACM Transactions on Modeling and Computer Simulation (TOMACS) July 1996 Volume 6 Issue 3

This article describes an approach to discrete event simulation modeling that appears to be effective for developing portable and efficient parallel execution of models of large distributed systems and communication networks. In this approach, the modeler develops submodels with an existing sequential simulation modeling tool, using the full expressive power of the tool. A set of modeling language extensions permits automatically sychronized communication between submodels; however, the aut ...

Results 1 - 2 of 2 **short listing**

Search Results

Search Results for: **[garbage<AND>((multiprocessor and heaps and synchronization and phases and memory))]**
Found **104** of **131,734 searched.**

Search within Results

GO > Advanced Search

> Search Help/Tips

Sort by: **Title** **Publication** **Publication Date** **Score** Binder

Results 1 - 20 of 104 **short listing**

Prev Page 1 2 3 4 5 6 Next Page

1 An on-the-fly mark and sweep garbage collector based on sliding views 97%
Hezi Azatchi , Yossi Levanoni , Harel Paz , Erez Petrank
ACM SIGPLAN Notices , Proceedings of the 18th ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications October 2003 Volume 38 Issue 11

With concurrent and garbage collected languages like Java and C# becoming popular, the need for a suitable non-intrusive, efficient, and concurrent multiprocessor garbage collector has become acute. We propose a novel mark and sweep on-the-fly algorithm based on the sliding views mechanism of Levanoni and Petrank. We have implemented our collector on the Jikes Java Virtual Machine running on a Netfinity multiprocessor and compared it to the concurrent algorithm and to the stop-the-world collecto ...

2 Portable, unobtrusive garbage collection for multiprocessor systems 94%
Damien Doligez , Georges Gonthier
Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages February 1994

We describe and prove the correctness of a new concurrent mark-and-sweep garbage collection algorithm. This algorithm derives from the classical on-the-fly algorithm from Dijkstra et al. [9]. A distinguishing feature of our algorithm is that it supports multiprocessor environments where the registers of running processes are not readily accessible, without imposing any overhead on the elementary operations of loading a register or reading or initializing a field. Furthermor ...

3 A parallel, real-time garbage collector 91%
Perry Cheng , Guy E. Blelloch
ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2001 conference on

Programming language design and implementation May 2001
Volume 36 Issue 5

We describe a parallel, real-time garbage collector and present experimental results that demonstrate good scalability and good real-time bounds. The collector is designed for shared-memory multiprocessors and is based on an earlier collector algorithm [2], which provided fixed bounds on the time any thread must pause for collection. However, since our earlier algorithm was designed for simple analysis, it had some impractical features. This paper presents the extensions necessary for a pract ...

4 Very concurrent mark-&-sweep garbage collection without fine-grain synchronization 91%

Lorenz Huelsbergen , Phil Winterbottom
ACM SIGPLAN Notices , Proceedings of the first international symposium on Memory management October 1998
Volume 34 Issue 3

We describe a new incremental algorithm for the concurrent reclamation of a program's allocated, yet unreachable, data. Our algorithm is a variant of mark-&-sweep collection that---unlike prior designs---runs mutator, marker, and sweeper threads concurrently *without* explicit fine-grain synchronization on shared-memory multiprocessors. A global, but infrequent, synchronization coordinates the per-object coloring marks used by the three threads; fine-grain synchronization is achieve ...

5 A parallel, incremental and concurrent GC for servers 90%

Yoav Ossia , Ori Ben-Yitzhak , Irit Goft , Elliot K. Kolodner , Victor Leikehman , Avi Owshanko
ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation May 2002
Volume 37 Issue 5

Multithreaded applications with multi-gigabyte heaps running on modern servers provide new challenges for garbage collection (GC). The challenges for "server-oriented" GC include: ensuring short pause times on a multi-gigabyte heap, while minimizing throughput penalty, good scaling on multiprocessor hardware, and keeping the number of expensive multi-cycle fence instructions required by weak ordering to a minimum. We designed and implemented a fully parallel, incremental, mostly concurrent colle ...

6 Garbage collecting the Internet: a survey of distributed garbage collection 89%

Saleh E. Abdullahi , Graem A. Ringwood
ACM Computing Surveys (CSUR) September 1998
Volume 30 Issue 3

Internet programming languages such as Java present new challenges to garbage-collection design. The spectrum of garbage-collection schema for linked structures distributed over a network are reviewed here. Distributed garbage collectors are classified first because they evolved from single-address-space collectors. This taxonomy is used as a framework to explore distribution issues: locality of action, communication overhead and indeterministic communication latency.

7 Eliminating synchronization bottlenecks using adaptive replication 88%

Martin C. Rinard , Pedro C. Diniz

ACM Transactions on Programming Languages and Systems (TOPLAS) May 2003
Volume 25 Issue 3

This article presents a new technique, adaptive replication, for automatically eliminating synchronization bottlenecks in multithreaded programs that perform atomic operations on objects. Synchronization bottlenecks occur when multiple threads attempt to concurrently update the same object. It is often possible to eliminate synchronization bottlenecks by replicating objects. Each thread can then update its own local replica without synchronization and without interacting with other threads. When ...

8 Sapphire: copying GC without stopping the world 87%

Richard L. Hudson , J. Eliot B. Moss
Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande June 2001

Many concurrent garbage collection (GC) algorithms have been devised, but few have been implemented and evaluated, particularly for the Java programming language. Sapphire is an algorithm we have devised for concurrent copying GC. Sapphire stresses minimizing the amount of time any given application thread may need to block to support the collector. In particular, Sapphire is intended to work well in the presence of a large number of application threads, on small- to medium-scale shared memor ...

9 A concurrent, generational garbage collector for a multithreaded implementation of ML 87%

Damien Doligez , Xavier Leroy
Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages March 1993

This paper presents the design and implementation of a "quasi real-time" garbage collector for Concurrent Caml Light, an implementation of ML with threads. This two-generation system combines a fast, asynchronous copying collector on the young generation with a non-disruptive concurrent marking collector on the old generation. This design crucially relies on the ML compile-time distinction between mutable and immutable objects.

10 Parallel execution of prolog programs: a survey 87%

Gopal Gupta , Enrico Pontelli , Khayri A.M. Ali , Mats Carlsson , Manuel V. Hermenegildo
ACM Transactions on Programming Languages and Systems (TOPLAS) July 2001
Volume 23 Issue 4

Since the early days of logic programming, researchers in the field realized the potential for exploitation of parallelism present in the execution of logic programs. Their high-level nature, the presence of nondeterminism, and their referential transparency, among other characteristics, make logic programs interesting candidates for obtaining speedups through parallel execution. At the same time, the fact that the typical applications of logic programming frequently involve irregular computatio ...

11 Implementing an on-the-fly garbage collector for Java 87%

Tamar Domani , Elliot K. Kolodner , Ethan Lewis , Eliot E. Salant , Katherine Barabash , Itai Lahan , Yossi Levanoni , Erez Petrank , Igor Yanorer
ACM SIGPLAN Notices , Proceedings of the second international symposium on Memory management October 2000
Volume 36 Issue 1

Java uses garbage collection (GC) for the automatic reclamation of computer memory no longer required by a running application. GC implementations for Java Virtual Machines (JVM) are typically designed for single processor machines, and do not necessarily perform well for a server program with many threads running on a multiprocessor. We designed and implemented an on-the-fly GC, based on the algorithm of Doligez, Leroy and Gonthier [13, 12] (DLG), for Java in this environment. An *on-the-f* ...

12 Supporting dynamic data structures on distributed-memory machines 85%

Anne Rogers , Martin C. Carlisle , John H. Reppy , Laurie J. Hendren

ACM Transactions on Programming Languages and Systems (TOPLAS) March 1995

Volume 17 Issue 2

Compiling for distributed-memory machines has been a very active research area in recent years. Much of this work has concentrated on programs that use arrays as their primary data structures. To date, little work has been done to address the problem of supporting programs that use pointer-based dynamic data structures. The techniques developed for supporting SPMD execution of array-based programs rely on the fact that arrays are statically defined and directly addressable. Recursive data s ...

13 The family of concurrent logic programming languages 85%

Ehud Shapiro

ACM Computing Surveys (CSUR) September 1989

Volume 21 Issue 3

Concurrent logic languages are high-level programming languages for parallel and distributed systems that offer a wide range of both known and novel concurrent programming techniques. Being logic programming languages, they preserve many advantages of the abstract logic programming model, including the logical reading of programs and computations, the convenience of representing data structures with logical terms and manipulating them using unification, and the amenability to metaprogrammin ...

14 Java without the coffee breaks: a nonintrusive multiprocessor garbage collector 85%

David F. Bacon , Clement R. Attanasio , Han B. Lee , V. T. Rajan , Stephen Smith

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation May 2001

Volume 36 Issue 5

The deployment of Java as a concurrent programming language has created a critical need for high-performance, concurrent, and incremental multiprocessor garbage collection. We present the *Recycler*, a fully concurrent pure reference counting garbage collector that we have implemented in the Jalapeño Java virtual machine running on shared memory multiprocessors.

While a variety of multiprocessor collectors have been proposed and some have been implemented, experimental dat ...

15 An on-the-fly reference counting garbage collector for Java 84%

Yossi Levanoni , Erez Petrank

ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications October 2001

Volume 36 Issue 11

Reference counting is not naturally suitable for running on multiprocessors. The update of pointers and reference counts requires atomic and synchronized operations. We present a novel reference counting algorithm suitable for a multiprocessor that does not require any synchronized operation in its write barrier (not even a compare-and-swap type of synchronization). The algorithm is efficient and may complete with any tracing algorithm.

16 Garbage collection for strongly-typed languages using run-time type reconstruction 84%

Shail Aditya , Christine H. Flood , James E. Hicks

ACM SIGPLAN Lisp Pointers , Proceedings of the 1994 ACM conference on LISP and functional programming July 1994

Volume VII Issue 3

Garbage collectors perform two functions: live-object detection and dead-object reclamation. In this paper, we present a new technique for live-object detection based on run-time type reconstruction for a strongly typed, polymorphic language. This scheme uses compile-time type information together with the run-time tree of activation frames to determine the exact type of every object participating in the computation. These reconstructed types are then used ...

17 Computing curricula 2001 83%

Journal on Educational Resources in Computing (JERIC) September 2001

18 A scalable mark-sweep garbage collector on large-scale shared-memory machines 83%

Toshio Endo , Kenjiro Taura , Akinori Yonezawa

Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM) November 1997

This work describes implementation of a mark-sweep garbage collector (GC) for shared-memory machines and reports its performance. It is a simple "parallel" collector in which all processors cooperatively traverse objects in the global shared heap. The collector stops the application program during a collection and assumes a uniform access cost to all locations in the shared heap. Implementation is based on the Boehm-Demers-Weiser conservative GC (Boehm GC). Experiments have been done on Ultra ...

19 Parcel: project for the automatic restructuring and concurrent evaluation of LISP 82%

L. Harrison

Proceedings of the 2nd international conference on Supercomputing June 1988

Parcel (Project for the Automatic Restructuring and Concurrent Evaluation of Lisp) is an investigation of the problem of compiling Lisp for evaluation on a shared memory multiprocessor. In this paper, we present an overview of the process of compilation in Parcel. This process consists, broadly, of an interprocedural analysis, followed by a function-level restructuring of the lambda expressions that constitute a program. We discuss both of these phases, and illustrate the steps of restructu ...

20 Thread-local heaps for Java 82%

Tamar Domani , Gal Goldshtein , Elliot K. Kolodner , Ethan Lewis , Erez Petrank , Dafna

Sheinwald
ACM SIGPLAN Notices , Proceedings of the third international symposium on Memory management June 2002
Volume 38 Issue 2 supplement

We present a memory management scheme for Java based on thread-local heaps. Assuming most objects are created and used by a single thread, it is desirable to free the memory manager from redundant synchronization for thread-local objects. Therefore, in our scheme each thread receives a partition of the heap in which it allocates its objects and in which it does local garbage collection without synchronization with other threads. We dynamically monitor to determine which objects are local and whi ...